AFRL-IF-RS-TR-2000-123
Final Technical Report
August 2000

THREE FINAL STEPS TOWARD PORTABILITY

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. F351

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

**AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK**

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2000-123 has been reviewed and is approved for publication.

APPROVED:

RALPH KOHLER
Project Engineer

FOR THE DIRECTOR:

NORTHRUP FOWLER, Technical Advisor
Information Technology Division
Information Directorate

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/IFTC, 26 Electronic Pky, Rome, NY 13441-4514. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

THREE FINAL STEPS TOWARD PORTABILITY

Craig Lund

Contractor: Mercury Computer Systems, Inc.
Contract Number: F30602-97-2-0271
Effective Date of Contract: 24 July 1997
Contract Expiration Date: 30 November 1999
Program Code Number: F351
Short Title of Work: Three Final Steps Toward Portability
Period of Work Covered: Jul 97 – Nov 99

Principal Investigator: Crain Lund
Phone: (508) 256-1300
AFRL Project Engineer: Ralph Kohler
Phone: (315) 330-2016

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense and was monitored by Ralph Kohler, AFRL/IFTC, 26 Electronic Pky, Rome, NY.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	AUGUST 2000	Final Jul 97 - Nov 99

4. TITLE AND SUBTITLE
THREE FINAL STEPS TOWARD PORTABILITY

5. FUNDING NUMBERS
C - F30602-97-2-0271
PE - 62301E
PR - D002
TA - 01
WU - P5

6. AUTHOR(S)
Craig Lund

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Mercury Computer Systems, Inc.
199 Riverneck Road
Chelmsford MA 01824-2820

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency
3701 N. Fairfax Drive
Arlington VA 22203-1714

Air Force Research Laboratory/IFTC
26 Electronic Pky
Rome NY 13441-4514

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2000-123

11. SUPPLEMENTARY NOTES
Air Force Research Laboratory Project Engineer: Ralph Kohler/IFTC/(315) 330-2016

12a. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
A software gap exists between the research and deployment communities. The research under this project is described by two research projects that build upon the Application Configuration language (ACL) foundation. The first is the "ACL" for Research Systems", which focused on code migration from research environments like NHPCC's into deployment situations. The underlying component programming model Mercury had advocated promotes the re-use of software modules and maintainability of large software projects. The second: "entering the Data Domain", created a standard API for real-time data shaping and data mapping - gluing the MPI and ACL worlds together in an innovative manner; eliminating the need for custom solutions. An industry standard data remapping API for signal processing has resulted from this effort. Recent progress towards standard programming tools and Application Programmer Interfaces (APIs) brings the embedded signal processing community closer to application source code portability. Portability will enable sharing code among signal processing projects and will allow applications to quickly take advantage of hardware from new vendors.

14. SUBJECT TERMS
Talaris, Scalability Challenges, Unix Launcher, RACE Generator Tool, Application Configuration Language, Automatic Mapper

15. NUMBER OF PAGES
56

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Standard Form 298 (Rev. 2-89) (EG)
Prescribed by ANSI Std. 239.18
Designed using Perform Pro, WHS/DIOR, Oct 94

Table of Contents

Appendixes

List of Figures

1.0 Executive Summary

This final report summarizes the results of research, that Mercury Computer Systems, Inc. MPI Software Technology (MSTI), the Albuquerque High Performance Computing Center (AHPCC), the Maui High Performance Computing Center (MHPCC), and the Naval Command, Control and Ocean Surveillance Center (NCCOSC) Research, Development, Test and Evaluation Division (SPAWAR) performed on the "Three Final Steps Toward Portability" program. The program was supported by the Defense Advanced Research Projects Agency (DARPA) under BAA 97-06, entitled "Adaptive Computing Systems, Embedded High Performance Computing and Ultrascale Computing".

A software gap exists between the research and deployment communities. Software created by researchers is rarely leveraged when primes contractors build real-time prototypes or deploy embedded systems. This is because the research and deployment communities use different tools, languages, and operating systems.

Recent progress towards standard programming tools and Application Programmer Interfaces (APIs) brings the embedded signal processing community closer to application source code portability. Portability will enable sharing code among signal processing projects. Portability will also allow applications to quickly take advantage of hardware from new vendors. The result will be faster development cycles, at lower cost, with more functionality.

Barriers to portability have continued to fall. One significant obstacle remains—mapping parallel software onto real hardware in a portable and productive way. Fortunately, Mercury Computer Systems, in collaboration with many commercial and academic organizations, has had a successful research program focused on productive, portable mapping techniques. The team members, as indicated above, have sought to push this research collaboration in two new directions. Our push attacked the last barriers to source code portability between real-time, deployment environments.

Our collaboration's past work in this area, centers on a language called "ACL"[1] for Application Configuration Language. ACL allows a programmer, or a programming tool, to specify a software structure, a hardware structure, and the mapping between them. The ACL language is general enough to incorporate arbitrary, legacy software—both unlinked modules and complete applications. Existing ACL tools such as The MathWorks' SIMULINK, Matra's CapCase, and Mercury's TCE allow users to describe software requirements, specific hardware configurations, and arbitrary mappings. In the hands of a programmer, the ACL language is powerful enough to go beyond expressing complex mapping algorithms. ACL can also capture fault recovery strategies, potentially allowing software to remap around hardware failures.

This report describes two research projects that build upon this ACL foundation. Each project was performed independently. Combined, the projects knocked down significant barriers to source code portability within parallel, real-time, deployment environments.

• The ***ACL for Research Systems*** project focused on code migration from research environments like MHPCC's into deployment situations (we needed this project if only to allow use of Mercury's previous MATLAB deployment effort within the MHPCC STAP community). The underlying component programming model Mercury has advocated promotes the re-use of software modules and maintainability of large software projects.

• The ***Entering the Data Domain*** project created a standard API for real-time data shaping and data mapping—gluing the MPI and ACL worlds together in an innovative manner; eliminating the need for custom solutions like Mercury's Parallel Application System (PAS) and SPAWAR's Scalable Parallel Environment (SPE). An industry standard data remapping API for signal processing has resulted from the (still continuing, independent from this contract) effort, again creating and stimulating improved code portability.

---

[1] For ACL overview materials, reference, and tutorial, see:

- www.mc.com/talaris_fold/talariseet.html
- www.mc.com/backgrounder_folder/icassp/icassp.html
- www.mc.com/talaris_fold/talaris/slide0.html

ACL for Research Systems

Mercury's implementation of ACL is named "Talaris" (Figure 1). Most of Talaris executes on a workstation. An portion of Talaris, called the "Generator", is specific to each run-time target. Generators create "Launch Kits". Most targets require small, stand-alone programs, called "Launchers", to execute these Launch Kits. A Generator and Launcher already exist for Mercury's MC/OS operating system on i860s, SHARCs, PowerPCs, as well as for Wind River's VxWorks on a 68000. At the start of this project no organization had yet elected to build a UNIX Generator and Launcher.

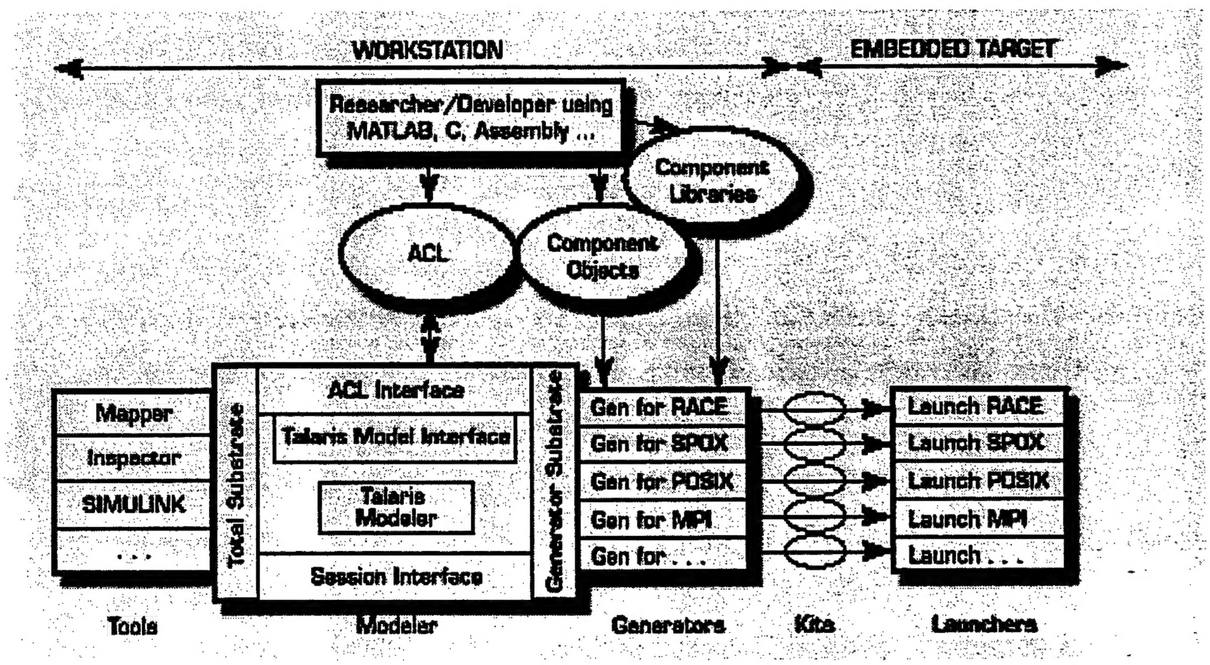


Figure 1. Talaris environment as a framework for component programming.

UNIX Shared-Memory Processor (SMP) boxes do not often encounter the scalability challenges that ACL was designed to meet. Scalability challenges refering to the problems associated with mapping an unchanging application between multiple hardware configurations in a deployment situation. Various defense platforms, for example, may be in different stages of an upgrade cycle, resulting in several hardware configurations that a single application must tolerate. Also, laboratory configurations very often differ from field hardware configurations. These are examples of problems ACL was designed to resolve.

UNIX boxes are a convenient place to experiment with code that will ultimately find its way into embedded environments. This fact made ACL interesting in such environments. MHPCC, for example, wanted to experiment with ACL in the UNIX research phase of projects with deployment potential.

There is yet another reason for creating a UNIX Launcher. Tool vendors, such as Khoral Research (KRI), target both embedded and traditional applications. ACL for UNIX would simplify creating such tools by eliminating a key difference between embedded and scientific platforms.

We actually required two different UNIX launchers, one variant targeting MPI middleware within interactive workstations clusters from Sun and IBM. The other targets MPI within the IBM-SP batch environment.

Entering the Data Domain

Existing DARPA investments in ACL and MPI represent a significant step towards portable signal processing software. Unfortunately, one critical API layer remains undefined. Signal processing programmers need an API that is focused on data "shape". Here we refer to the patterns that programmers use to scatter data sets between memory systems within a distributed, parallel, hardware environment. Signal processing programmers need an API that can "transpose" data rows and columns, not just a message passing API.

Visual signal processing tools also require such an API. Like programmers, visual tools that target signal processing "think" in terms of data flow and shape, a model MPI and MPI/RT are not optimized for.

Our developed Data Reorganization Interface (DRI) API is something ideally layered "above" MPI/RT. However, any runtime that supports a shape-oriented API faces a significant algorithm/software/hardware mapping challenge — ACL's domain.

Thus our team wanted to define a data shape/remapping API that can leverage both MPI/RT and ACL. Our team wanted to cooperatively study the issue. Mercury intends to promote the team's conclusions throughout the DoD signal processing community.

Our team had planned to leverage the development of a MPI/RT specification for "transpose and reshape," whose purpose was to introduce an early-binding form of data reorganization into MPI/RT. Participants have pushed for the addition of "early binding, collective redistribution" into MPI.

Our team has not asked DARPA, during the program or at this time, for help building the first implementation of our new API. This is because, since we were successful at building consensus, we believe that platform suppliers like Mercury will create implementations at their own expense.

Conclusion

A third element of the originally proposed program, as indicated in the project's name was unfortunately not funded under this effort. In it, as a continuation of the DARPA BAA 95-19 Mercury's "Bridging the Development Gap" contract, our team was to define an API specification for mapping algorithms, and implement a tool that can call these interchangeable mapping algorithms to accomplish mapping. The software and hardware specification for this tool was to be ACL. KRI was originally proposed to build the Automatic Mapper as an enhancement to Mercury's TCE.

The completed and reported effort has shrunk the percieved software gap by creating standards and a tool infrastructure that the research and deployment communities can share. Our work has been intensitively focused on parallel programming and on multicomputer data mapping challenges specific to signal processing.

2.0 Introduction

This final report describes recent research and development work related to BAA 97-06 that has significantly improved developer productivity for parallel programming of signal processing applications today, while laying the groundwork for dramatic advances in the future.

Mercury's ACL is a language that allows programmers to treat legacy software object libraries as software components with defined interfaces (stack frames, POSIX sockets, MPI/RT channels, and so on). Using ACL, programmers can express how they wish to interconnect software components into a complete application. ACL can capture similar information on target hardware configurations. The real power of ACL is its ability to then express how to map a software concept into a hardware reality. Because ACL is a complete programming language based upon the popular TCL, such mappings can be algorithms that react to change when, for example, hardware fails.

Mercury's TCE is a tool "framework" with plug in extensions that manipulates ACL. TCE, developed under a previous DARPA contract, runs on workstations. Team members have implemented many of the project elements as TCE extensions.

ACL for Research Systems

An ACL environment already exists for Mercury's MC/OS environment and for Wind River's VxWorks. Spectron is working to add ACL support to its SPOX-MP. Extending Mercury's tool suite to support each new target requires that someone develop a new "Generator" for each target. Most environments also need a corresponding "Launcher" which executes on the targets (see Figure 1). Mercury is to implement the new Generators as extensions to TCE.

Generators emit "Launch Kits." A Generator understands its target's capabilities (such as shared memory and sockets), the target's operating system services, and the target's development tool chain.

Launchers processes Launch Kits at runtime. Launch Kits contain a description of which object files a Launcher must load onto each target processor. Kits also describe resource requirements such as interprocess communication objects that Launchers must establish before handing control over to a user's application.

Developing a Generator and Launcher for each new target requires tackling several challenges that are not immediately obvious. First among them is that UNIX or MPP platforms are rarely dedicated to a single application. Therefore, the ACL Launcher will not have control of the entire platform. Our new Launchers will need to request resources (processors, memory, or workstations) from a higher authority, a concept not built into original Launcher implementations. We have developed Launchers and Generators for three new target environments:

- ***POSIX Reference Implementation.*** This provides a UNIX reference implementation of ACL that leverages the POSIX API (i. e. shared memory, semaphores, sockets, threads) running on an SMP SPARCstation using Solaris 2.x. From this starting point other organizations can create ports to other POSIX environments.
- ***MPI Reference Implementation.*** This provides an MPI reference implementation based on a cluster of Sun Workstations using MPICH. This reference implementation provides a starting point for future MPI-based implementations of ACL.
- ***IBM SP-2 Implementation.*** Starting with the MPI Reference Implementation described above, MHPCC has provided enhancements to support an IBM SP-2 using IBM's MPI.

We have made these enhancements by changing the Talaris tool chain in a significant way. Today Talaris has a single "Generator" component which creates the "Launch Kits" that Launchers execute. Mercury's experience with Spectron's ACL work has shown that a better approach is to have a separate generator for each target environment (MC/OS, VxWorks, SPOX-MP, POSIX, MPI). After consulting with Mercury, Spectron had suggested that Mercury expose the toolbox inside the TCE that manipulates ACL. Mercury calls this Java toolbox the Configuration Model Interface (CMI). Mercury has

adopted this approach. We first created a generic generator designed to supplant our existing generator. We then created two variants, one for POSIX and another for MPI. Because of the way in which TCE was written in the Java programming language, the new generators became Java Beans. (An introduction to Java Beans technology can be found at <http://splash.javasoft.com/beans/WhitePaper.html>).

Entering the Data Domain

Our team did not intend to define a data shaping/remapping API without focusing much of our time initially on the runtime system that sits underneath the API. This is because, when a program requests a data remap, the runtime system must query the software/hardware map to determine how many memory systems are involved. This fact implies that significant data movement decisions must be made after compile and link time. Making decisions this late is a challenge because "early binding" is a key foundation of performance in all real-time systems. The most flexible solution requires making all data movement strategy decisions at runtime. The highest performance solution requires making such decisions just after the map is known, but before execution begins.

At the start of the program existing APIs do not allow the latter approach. Existing data shape/remapping APIs focussed on either flexibility or performance, but rarely on both. KRI's Distributed Data Services was one extreme example. It is very flexible at run time, but there is no hope of ever fitting it into a DSP chip's high performance, restricted resource environment. Mercury's PAS was at the opposite extreme: a very targeted feature set that fits into a SHARC, but with little runtime flexibility. Neither example integrates well with MPI.

The community needed something better. In this program we studied high performance options and then publicly proposed solutions to the embedded community. The team will declare success only after we have build broad consensus in the community.

3.0 Mercury Computer Systems Program Participation

In Mercury's approach to component programming, a software application is expressed as an interconnection of software Modules that executes on a configuration of hardware Modules. A software Module consists of executable code that operates on data and commands via one or more Ports of the Modules. Interconnections of Ports between Modules are Connections. Graphically, the relationships of Modules, Ports, and Connections are shown in Figure 2.

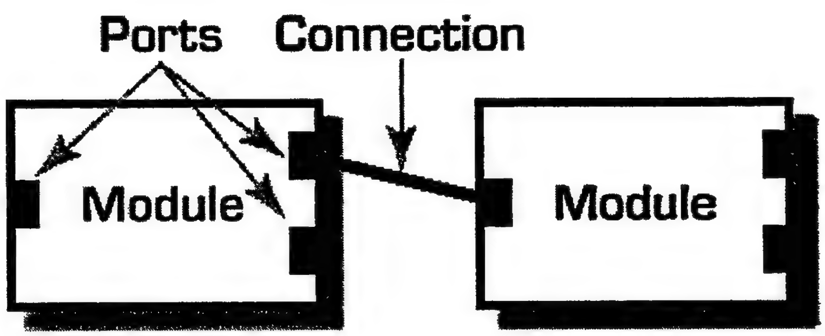


Figure 2. Representation of Modules, Ports and Connections

In the RACE implementation, Modules are POSIX threads or processes, Ports are various types of protocols (e.g., message passing, synchronization, and shared-memory application programming interfaces (APIs)), and Connections are objects that attach to Ports. Hardware Modules consist of processors and their memory systems, the interface to the processor, and the connection of interfaces (e.g., connection to a shared bus or point-to-point fabric).

ACL

Mercury has developed an underlying run-time system that supports a component programming model. Such a run-time system processes a "netlist" which specifies the interconnection and processor assignments of software modules available as object code. From the netlist, the runtime system synthesizes the required executable images, loads the images into appropriate processors, sets up the "interconnections" as inter-process communication objects, and begins execution of the application. The underlying "netlist"

specification is actually a scripting language. Specifically, we have created a specialized Tool Command Language (TCL) extension package that we call ACL (Application Configuration Language).

Talaris

Mercury has also created a powerful environment for cooperating tools that better support the development of complex applications. Implemented in Java, the Talaris Environment is very portable and extensible. It is being applied to an increasing range of target systems and programming interfaces.

Talaris Modeler -- The Model is controlled by a transaction engine called the *Talaris Modeler* that manages and synchronizes (in real-time) both the Model *and* the corresponding ACL command source. The Modeler is equally at ease with the ACL source form of the Model and the underlying Model itself, with its computed entities and relationships. Because the Modeler supports a Model-View-Controller architecture, all views of the Model – including the ACL command form – are constantly synchronized as the contents of the Model are changed.

Talaris Tools -- On startup, the Environment uses initialization settings to establish the semantics of the Model, the number and names of Model domains, the initial type hierarchy, and the selected target system. The developer proceeds to open, edit, and save ACL documents with Editing Tools. When the configuration is complete, the developer uses Target Tools to build and launch the application on a specific target system.

Mercury's RACE Generator Tool -- To create applications to launch on RACE systems, Mercury's RACE Generator Tool builds a *launch kit* in a sequence of four phases:

- Analysis – scan the Model, resolve ambiguities and other defaults, perform initialization deadlock analysis
- Files – create all the input to the launch kit, initialization instructions, dependency file, dispatch tables

- Build – compile dispatch tables, build the executable images
- Kit – create a launch kit from the output of the files and build phases. To reduce the time required to create a launch kit, the Generator computes the minimum build plan. The analysis and files phases are done only if the model has changed, the build phase is done only if its inputs have changed, and the kit phase updates the final kit contents only if the build phase resulted in changes. This enables fast turnaround during kit generation as well as during launch, because redundant image-loads are thereby avoided.

Mercury's RACE Launcher -- Mercury's RACE Launcher starts applications on RACE systems in four steps:

- **Image load.** Note that a sizeable application may involve several hundred megabytes of executable images being loaded onto hundreds of processors. Using features of MC/OS, the RACE Launcher avoids reloading images that have not changed since the last launch.
- **Process and thread creation.** The Launcher constructs argc/argv/env data for processes and marshals arguments for thread entry points.
- **Initialization.** To start properly and without deadlock, a complex application requires precise setup of thousands of software communication mechanisms. The Launcher is able to do this in coordinated phases, avoiding possible deadlock conditions in a highly orchestrated communication protocol between the Launcher and small *agent modules* that have been placed in each process by the Generator.
- **Initiation.** With all initialization complete, the Launcher releases threads and starts processes.

With this infrastructure, application development is equated to building a fully specified and populated application model in the Talaris Modeler. A fully specified application model contains:

- A system hardware model that expresses the instances of hardware Modules and their interconnection.

- A software model that expresses the instances of software Modules and their interconnection.
- The assignment of software Modules to hardware Modules.

A fully populated model means that object files (i.e., a ".o" file or library entry) exist for each software Module (for the assigned hardware Module type) and the hardware exists.

Program Accomplishments

Mercury has modified the Launcher from its initial ACL tool suite. Mercury has also modified the initial Talaris "Generator" scheme to allow targeting the new Launchers. The resulting derivative work leveraged into POSIX and MPICH running over a network of Sun workstations. The following steps have achieved this goal:

- Mercury has written a detailed project plan that described how Mercury planned to extend its initial ACL tool suite to support UNIX workstation clusters running POSIX and MPICH.
- Mercury has organized a meeting of a subset of the team to review the project plan. Mercury has updated its plan to reflect comments from team members.
- Mercury has made source code modifications as called for in the plan and written documentation. The result was an "alpha" quality binary release that Mercury has made available upon request. For the next six months Mercury accepted, and acted upon, bug reports received from users. Mercury has made multiple binary releases during this period. At completion, Mercury has elevated the release's quality label from "alpha" to "beta". A CD-ROM distribution is available upon request.

After the Sun binary release reaches beta quality, Mercury has turned all source code we believed necessary over to MHPCC. MHPCC has used that to port the source code to its IBM SP-2 which runs IBM's MPI (see next Section).

4.0 MHPCC Program Participation

Existing Mercury and DARPA investments in software tools for embeddable computing have resulted in substantial improvement in the portability between Mercury embedded solutions and UNIX based architectures. A complete porting solution involves the integration of a number of standardized software components such as the Message Passing Interface (MPI and MPI/RT), Vector Signal Image Processing Library (VSIPL), and the Application Configuration Language (ACL). MHPCC has been supporting Mercury's efforts to provide a transition path by supporting the development of an Application Programmer Interface (API) through the *Entering the Data Domain* task and through the port of the Application Configuration Language (ACL) to the IBM SP architecture.

The Application Configuration Language (ACL) is a mechanism for describing the interconnection of software components on embeddable hardware utilizing an abstraction that isolates computational software modules from the details of hardware interconnection topology. ACL facilitates the utilization of legacy software libraries in multiprocessor architectures by providing a reconfigurable mechanism for mapping software processes onto hardware structure.

Program Accomplishments

Talaris/ACL was successfully migrated to an IBM workstation at AHPCC and to multiple interactive IBM-SP nodes at MHPCC. The Talaris/ACL port is fully functional using MPICH/P4 on the Ethernet communication fabric. A number of Talaris example programs have been successfully compiled, generated, and launched on both an IBM RS6000 workstation at AHPCC, and on multiple interactive IBM-SP nodes at MHPCC. These examples utilize a variety of transport types including System V shared memory and semaphores, sockets, and MPI.

A number of modifications to the launcher, agent, and generator code were required in order to complete the Talaris/ACL port to the AIX environment. Changes were also

made to the ACL files for the Talaris example programs in order to make them work under AIX. These modifications are summarized in the following sections.

Talaris Modifications for SP2

Launcher and Agent -- The few problems encountered in migration of the launcher and agent C-language code to the AIX environment were difficult and time-consuming. One problem arose from the fact that the order of stack evaluation for the C-language is not specified in any standard. Thus, procedure calls with formal arguments of the form

```
status = pthread_create(&array[i++], &value, &array[i]);
```

can produce different results depending on whether the formal arguments are evaluated left-to-right or right-to-left. The solution was to remove the order of stack evaluation dependency from procedure calls in the agent and launcher code.

A second problem arose from the fact that Solaris 2.6 pthreads are based on the Version 10 standard whereas AIX 4.2 pthreads are based on the Version 7 standard. The default behavior for Version 10 pthread creation is to make threads non-detachable (joinable), whereas the default behavior for Version 7 pthread creation is to make threads detachable (non-joinable). Talaris/ACL pthreads created by the launcher and agent are assumed to be joinable, which is not the default behavior of pthread creation under AIX 4.2. The solution was to issue a call, after pthread attribute initialization, to explicitly set the pthread detach-state.

The last problem arose from the fact that the AIX 4.2 *sockaddr* struct type is slightly different in content and size, from the BSD4.3-compatible Solaris sockaddr struct. This difference was causing the agent socket code to truncate socket pathnames by a single character. The system error message produced by this was a very misleading "error - socket address in use". The solution was to specify a switch –DCOMPAT_43 on the agent compiler line, which instructed the compiler to utilize a BSD4.3-compatible sockaddr struct.

Once these three final changes were made, the launcher and agent code began to work under the AIX operating system on both RS6000 workstation and IBM-SP nodes.

Generator -- At present, the existing generator is being coerced into producing AIX compatible executables by overriding the behaviors associated with the native compute node type *SOLARIS_NODE*. This was accomplished by:

(1) Adding the statements

    set_property SOLARIS_NODE -compiler_tool xlc_r
    set_property SOLARIS_NODE -linker_tool xlc_r

to the global util.tcl file in the example program directory.

(2) Establishing the following soft-links in the /usr/mercury/talaris/utp/lib directory

    libpthread.a -> /usr/lib/libpthreads.a
    libposix4.a -> /usr/lib/libxnet_r.a
    libnsl.a-> /usr/lib/libnsl_r.a
    libsocket.a -> /usr/lib/libc_r.a

so that the generator linker line for the SOLARIS_NODE type

    $(CC) .... -lagent -lpthread -lposix4 -lnsl -lsocket

would, in effect, produce the equivalent of the following for AIX

    $(CC) .... –lagent -lpthreads -lxnet_r -lnsl_r -lc_r

(3) Establishing the following soft-link in the /usr/mercury/talaris/utp/lib directory

    libmpi.a -> /usr/local/mpich/..../rs6000/lib/libmpich.a

so that user applications using MPI will link the MPICH/P4 library.

A new node type, *AIX_NODE*, was introduced to the Talaris Unix Target Package (UTP). The changes introduced to the Java code will require a compile/rebuild of the Talaris generator. Because of the loss of key developers and the lack of any previous Makefile to use as a template, rebuild and test of the Java code changes was not completed. Once completed, user applications will only need to specify the AIX_NODE type in their ACL

code in order to generate AIX-compatible code. Thus, the temporary workarounds involving SOLARIS_NODE overrides will not longer be needed.

Example Program Common Makefile and ACL File Modifications -- Minor modifications were made to the *common.mk* make dependency file in the Talaris examples directory to make use of the thread-safe compiler and pull in the appropriate header files for the MPICH/P4 libraries. For reference, the CFLAGS were defined as follows:

```
CFLAGS = -g –qchars=signed –qfold –qlangvl=ansi \
         -D_ALL_SOURCE –D_XOPEN_SOURCE_EXTENDED=1 \
         -D_COMPAT_43 –DAIX –DOSV=42 –I/usr/local/mpich/include
```

The mpich include path, specified here with the –I switch, may be different at different sites.

The individual example ACL_TCL files were modified to utilize transport pathnames compatible with the local AIX environment. That is, transport pathnames of the form */UTPt00bsoc*, that require root permission under AIX, were replaced by pathnames of the form */nfs/sigpro/b* at AHPCC, and by pathnames of the form */scratch4/t00b* at MHPCC. The pathnames must refer to nfs-mounted partitions, for accessibility by multiple processors. The pathnames were limited to 14 characters for maximum compatibility across operating systems.

Finally, for the examples (t08 through t13) that utilize MPI, the following conditional was added to accommodate a missing define under AIX.

```
#ifndef MPI_CHARACTER
#define MPI_CHAR  MPI_CHARACTER
#endif
```

Talaris Installation on the SP2

The alpha release of Talaris for the SP2 is set up using the Solaris version installation, which is then modified using an instruction sheet for the SP2 environment. At such time as the Generator modifications to support the AIX_NODE flag are compiled and tested, references to "borrowing" the SOLARIS_NODE definitions will be deleted.

Talaris Limitations on the SP2 -- The alpha release of Talaris for the SP is configured to use MPICH 1.1 or 1.2, rather than the native AIX MPI implementation. The test cases have not yet been run with the compiler flags set to use the fast switch in user space. This results in inter-node communication running over Ethernet in IP space, which has considerably lower bandwidth.

Talaris is unable to run jobs in batch mode. Talaris replicates much of the functionality of IBM's SP2 Parallel Operating Environment (POE). The POE consists of a set of software components for developing, compiling, executing, debugging, profiling, and tuning parallel programs. A typical SP user will have a number of POE environment variables set via C shell scripts, none of which are applicable to Talaris. In addition, both IBM's Load Leveler and MHPCC's Maui Scheduler work with POE to control the batch job queues. Preliminary investigation concluded that modifying Talaris to support batch mode would require a significant development project on its own, and likely necessitate disabling POE and Load Leveler.

Presentation at Embedded Processing Principal Investigator Meeting at MHPCC

Joe Fogler of AHPCC, Karen Lauro and Henk Spaanenburg of Mercury during a DARPA-sponsored Embedded PI meeting in Maui the week of March 15, 1999 assisted with the installation and configuration of a Talaris 2.1 Beta demo on a pair of SPARC machines running Solaris 2.6 at MHPCC (see Appendix A for the presentation).

The results of the MHPCC work are now available on their internal web site, including on-line documentation, source code, and executables.

5.0 MSTI Program Participation

The "Entering the Data Domain" effort under DARPA support of the Mercury "Three Final Steps to Portability" program has achieved significant results and has focused community effort on Data Reorganization Interface (DRI) issues. This effort is concerned with the issue of data reorganization for datacubes, and has specifically created the "DARPA Data Reorganization Forum," with attendant standards draft document[2], informative web pages, and has met regularly during the program[3].

MPI Issues

The original MPI functions for "all-to-all" were unable to handle general, non-square single-group remapping, and omitted dataflow remapping. MPI-2 corrected this in part, but failed to provide early binding. MPI/RT ("MPI/RT – An Emerging Standard for High-Performance Real-Time Systems", A. Kanevsky, A. Skjellum, A. Rounbehler, Proceedings of the 31st Annual Hawaii International Conference on System Sciences, 1998) corrected the lack of early binding. However, MPI/RT's definition also recognized that the level of abstraction needed for efficient transpose and reshape needed a different API. This API would offer the interface for MPI/RT programmers to get the benefits of early binding. However, none of the benefits of mapping are guaranteed. This has been studied from two respects. First, the quality of service specifications for all MPI/RT collective operations can conceivably be extended to include mapping hints. Second, the types of quality of service offered for MPI/RT collective operations can be augmented appropriately. In addition, the notion of adding quality of service or other constraints to the spawning commands of MPI/RT (drawn from MPI-2) can be used to help guide system choices that will later enable quality of service to be realized.

Documentation of the type of transpose and reshape extensions appropriate to *Entering the Data Domain* has been undertaken. Furthermore, the efficacy of the "MPI/RT

---

[2] Data Reorganization Interface Bindings, MPI Binding Specification, and Draft DARPA Data Reorganization Development document.

[3] Meetings in Boston, MA (05-Nov-97), Albuquerque, NM (13-Jan-98), Boston, MA (05-Mar-98), Starkville, MS (21-May-98), Boston, MA (25-Sep-98), Bedford, MA (01-Dec-98), San Diego, CA (02-Feb-99), Moorestown, NJ (11-Jun-99)

datacube reshape" has been reviewed by the forum participants, and common application use scenarios have been connected with the interface, and have been graded for efficacy and ease-of-use.

The general permutation mappings of the MPI/RT transpose and reshape has been considered for application situations well known to KRI, Mercury, and SPAWAR, and MSTI. MSTI has considered possible generalizations to these permutations, when and if appropriate.

Entering the Data Domain

Application programs that perform data parallel operations for signal processing require data flow between memory systems within a distributed, parallel hardware environment. Often, successive algorithm stages require the reorganization of data as it flows in pipelined fashion between processing elements. The Message Passing Interface provides a unified API for the simple movement of data but does not address the concept of data shape and the need for reorganization of data shape.

The purpose of the Entering the Data Domain task was to address a critical need for a unified Application Programmer Interface (API) that describes software objects and methods for the movement and reorganization of distributed data shapes. This API defines a software layer that builds upon existing interprocess communication (IPC) middleware such as MPI, and provides a higher-level interface to simplify the development of parallel distributed signal processing applications.

Although MPI is sometimes viewed as the focus of development, the data reorganization interface API being developed is actually much broader in scope and is intended to serve as a guide for software development involving other IPC middleware including MPI/RT, VIA, PAS, and others. MSTI plans to implement the *Entering the Data Domain* extensions as part of its commercial MPI offerings.

Program Accomplishments

MSTI undertook a series of standardization meetings as specified by the statement of work, to bring together members of the embedded and parallel processing community in order to define and standardize a set of procedures (syntax and semantics) for data-reorganization of datacubes. This set of services is a key technology for in-place and out-of-place parallel computations for important classes of signal and image processing of relevance to US DOD and other areas, such as medical imaging.

Key accomplishments are as follows: development of a meta-API specification for describing important data-reorganization primitives, extensive discussions on how to integrate these with MPI-1.2 and MPI/RT, and an expansion of research and practical knowledge concerning the state of the art in expressive primitives for data reorganization. The documents describing this work, together with the minutes of the meetings, are posted at , www.data-re.org; this site will be kept active indefinitely by MSTI in order to continue promotion of the area.

The documents as currently developed offer an initial set of meta-API instructions that can be specialized for a number of message passing systems. Specific proposals have been made thus far on how to do this specialization for MPI-1.2 (as extensions). Approaches for doing the same type of specialization for MPI/RT have not been developed, but are closely related to the approaches to be followed for MPI-1.2 extensions, with certain exceptions that make sense to hash out within the MPI/RT Forum, rather than within the Data Reorganization Forum.

The body of knowledge developed addresses a low-level API, one that works with objects and permutations on index sets for tactile matrices and vectors. It has been determined through the course of this research effort that a competing, high-level tensor approach is also possible, either as a complement or in lieu of the low-level implementation. It has generally been agreed that the tensor approach is a good topic for future research, and needs to be strongly correlated with any parallel extensions for VSIPL. Literature in the area of multi-dimensional FFTs suggests that vector, parallel, and superscalar multi-

dimensional FFTs must be closely linked to the data reorganization problem in order to achieve optimal performance (e.g., Tolimieri 1997). In this sense, the committee has restricted itself to the low-level meta-API, recognizing that much more work is needed to achieve a high-level meta-API as well.

The low-level meta-API is capable of describing in place and bi-partite corner turn operations for N-dimensional dense data objects, where N is limited to six-dimensional in the current draft. This arbitrary limitation was done based on expediency and perceived application needs at this time. The data distributions and conformations supported reflect both explicit and implicit indexing strategies commonly in use in the High Performance Computing community.

The documents as currently developed are supplemented by a set of minutes and proposals that form the entirety of the results thus far. and together form the body of knowledge created. Because competing ideas remain about instantiation of the meta-API, this remains work for the future.

Despite completion of this DARPA program's tasks, this effort is going to continue on a pro bono effort for at least an additional year, held in conjunction with MPI/RT and VSIPL meetings. The purpose of the continued meetings are as follows: to see specific instantiations of the meta-API introduced into a future revision of the MPI/RT standard, and to explore relationships with any parallel VSIPL that emerges (since data reorganization and parallel FFTs are strongly correlated). These efforts are beyond the scope of the orginal task descriptions under this contract, but are being undertaken by a set of volunteers because of the perceived extreme value of the Data Reorganization effort to the High Performance Embedded Community.

The working group was chaired initially by Anthony Skjellum, who later added Ken Cain of MITRE, as co-chair, given his strong involvement. Other key contributors are Jon Greene (Mercury Computer Systems), James Lebak (MIT Lincoln Labs), and Nathan Doss (Lockheed Martin GES).

6.0 Summary and Future Research

The following sections summarize our findings and products, and also introduces new elements in the programming environment for portability.

ACL for Research Systems

Mercury will make the new generators and launchers available to anyone who asks, at no cost (with the possible exception of reasonable media and administrative fees). The generators are extensions to TCE (TCE is available thanks to a previous DARPA contract). The new Launchers are small programs specific to the platform they target (i. e. Sun MPICH and IBM MPI). This proposal, combined with Mercury's previous DARPA contract, makes large portions of TCE, and a portable implementation of ACL, available to anyone who asks, at no cost beyond potential media and administrative charges.

However, an important part of both TCE and Portable ACL remains untouched by DARPA investment. We call this item the "Modeler." It is the core of both tools and represents a significant Mercury investment. We offer access to the Modeler as an in-kind, cost sharing contribution. This means that Mercury will make Modeler binaries available to anyone who asks, providing that person is engaged in non profit, DARPA-sponsored research. Mercury may demand a reasonable royalty from people who plan to commercialize our Modeler code.

Operating system vendors can use this code to quickly incorporate ACL functionality into their offerings. In addition, tool vendors may use this source code to layer ACL functionality onto operating systems that do not offer ACL.

MHPCC could have delivered the IBM SP-2 generator and launcher to KRI where it could have been combined with the output of the originally proposed, but not funded by DARPA, *An Automatic Mapper* project into a mapping and launching tool suite. MHPCC would have coordinated and established acceptance criteria with KRI to insure the port meets KRI's software standards and conforms to KRI's software environment.

An Automatic Mapper

The Automatic Mapper task has been proposed to provide an extensible framework in which to develop and characterize algorithms for mapping software systems onto hardware.

Mapping can be described most succinctly as a process in which a software system is mapped onto available hardware such that a performance-motivated objective function is optimized. Algorithm mapping for embedded systems is a non-linear optimization problem that depends on memory consumption and communication latency among other factors. Automatic mapping employs the user of an algorithm to perform the mapping without manual intervention. The appeal of automatic mapping is that it eliminates an often-repeated, time-consuming task from the algorithm developer and thus reduces cost and increases the portability of the algorithm across multiple hardware configurations

Automatic mapping is a current research topic. Automatic mapping algorithms have been published from Ptolemy-related research at the University of California, Berkeley and the PARSA project at the University of Texas, Arlington. This effort would not attempt to discover new mapping algorithms. We propose to produce a framework to allow flexible algorithm selection for the mapping process. The resulting framework will allow different mapping algorithms to be rapidly introduced and applied to software and hardware to be mapped. Reference mapping algorithms will be implemented to verify the framework's utility.

Such a mapper starts with an algorithm represented as a graph in which nodes represent computational steps and arcs represent data or control flow, and a hardware description in which nodes represent computational elements (processors) and arcs represent data pathways. Additional information representing operational parameters will be attached to each node and arc to describe the performance or behavioral characteristics of each. Software parameters include computational requirements, data size and shape information, and similar. Hardware parameters include performance characteristics of processing elements as well as hardware limitations such as memory capacity, and

communication fabric parameters such as bandwidth. latency, and routing characteristics. The proposed automatic mapper will accept these two graphs as input and produce a mapping of the software onto the hardware that meets the constraints imposed by each. The goal of such a mapper is to produce an optimal mapping. Clearly, producing an optimal map is an intractable problem. However, a sub-optimal solutions can be achieved by using first-fit, simulated annealing and greedy algorithm models.

The intractable nature of this problem argues strongly for the availability of a framework that allows rapid insertion of mapping algorithms for evaluation purposes. The utility of such a framework is to provide a means for researchers to test and characterize heuristic mapping algorithms to determine their viability without being required to implement new testing harnesses for each mapping algorithm. Such a framework would include a description syntax for hardware and software graphs including hardware and software performance data as well as an API for graph interrogation.

As input, the framework will process ACL descriptions of both hardware and software. Hardware and software performance data will initially be imported as separate, supporting information using a syntax to be determined. Later, KRI and Mercury will consider using ACL "properties" to carry hardware and software performance data.

KRI is interested in the "Entering the Data Domain" study because KRI faces an identical challenge within its Embedded Khoros effort. As we have previously stated, Embedded Khoros cannot afford the overhead associated with Distributed Data Services. Therefore, KRI must define a Data Services subset that can meet embedded performance constraints.

Entering the Data Domain

The result of this research program is a public domain standards document. We hope that commercial vendors will implement the standard at their own expense. We believe that vendor implementations will follow at no cost to DARPA.

Appendix A.

Final presentation and demonstration at DARPA Embedded Systems PI meeting

(PowerPoint Presentation)

Talaris Application Development Environment for Scalable Embedded Computers

Embedded Systems Principal Investigator Meeting
Maui, Hawaii, March 1999

(BAA97-07 : Contract F30602-97-2-0271)

Karen Lauro
klauro@mc.com

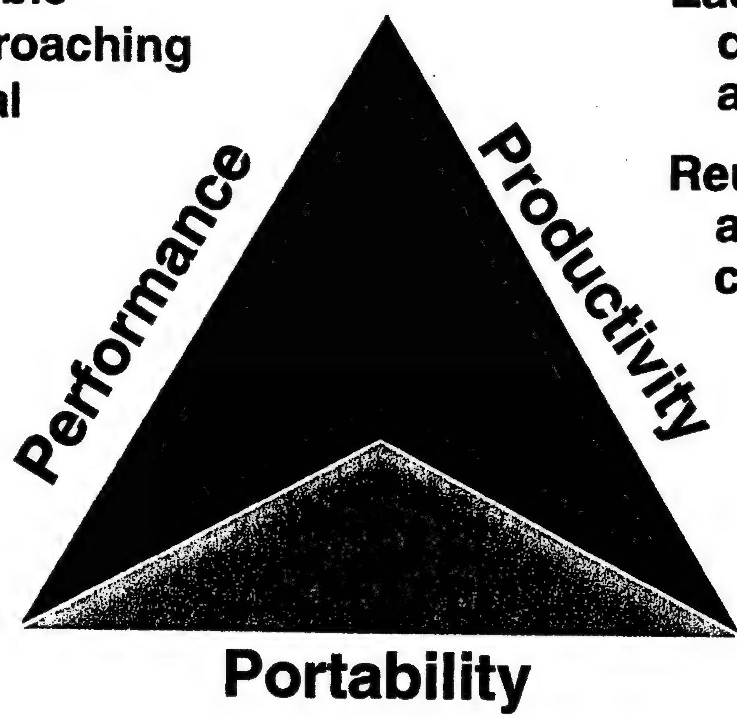
Mercury Computer Systems, Inc.

Application Development Objectives:
System Affordability

Application obtainable performance approaching system theoretical performance.

Ease of programming, debugging, optimization, and maintenance.

Reusablity of software and configuration components.



Workstation development and testing of algorithms for deployment on embedded target systems

Motivation for Talaris Development

Talaris was developed to address four main deficiencies in traditional application development:

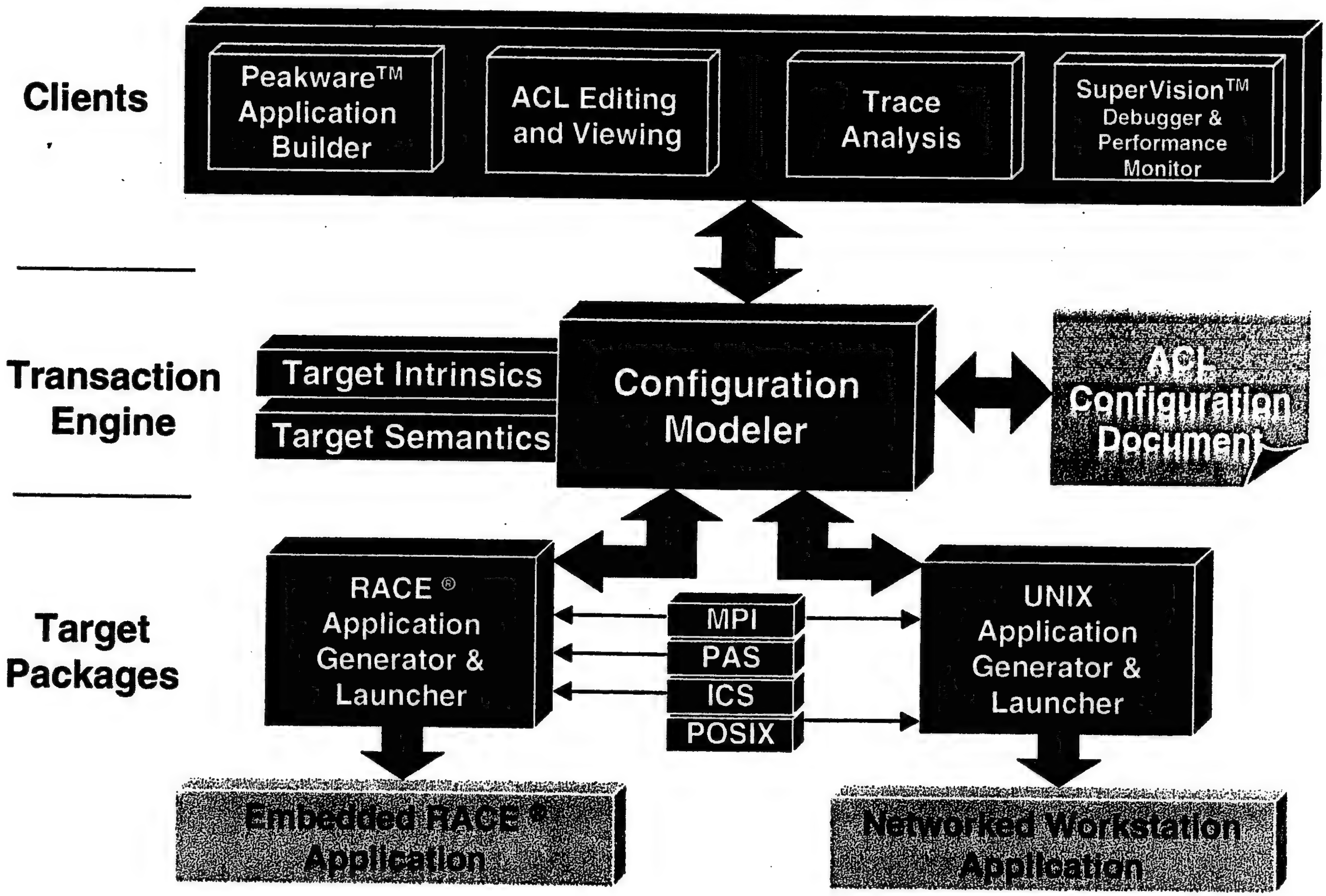
- The topographic complexity of distributed applications can not be efficiently relayed in distributed sequential source code.
- Too much effort is being expended on writing platform-specific interprocessor communication code relative to the algorithmic code, at the expense of time-to-market and portability.
- A component's system knowledge should be limited to the processes it communicates directly with.
- Properties of an integrated system are more amenable to monitoring and analysis. System information sharing allows the development of powerful tools.

Talaris™ Three Tier Software Architecture

The Talaris™ is a component-based application environment for large-scale embedded signal and image processing applications.

- Central to the infrastructure is the configuration modeling transaction engine.
  - The configuration model can be dynamically extended with Model Semantics and Intrinsics Plug-ins, tailoring it for a given type of system, application, or development tool, and allowing portability of applications across different hardware and software platforms.
  - Talaris constantly maintains a human readable view of the model is the form of an Application Configuration Language (ACL) document. The ACL script provides model persistence and can be saved and shared as a readable text file commands.
- Upon this extensible infrastructure are Talaris's expandable client tools which interface to the engine to edit, interact, and process the application configuration model.
- In Mercury's initial implementation, the clients encompass the system design, implementation, configuration, debug and optimization phases of application life cycle.

Talaris™ Three Tier Software Architecture



Talaris™ Portability

- Application Configuration Language (ACL) supports customized model semantics and intrinsics per target system
- ACL is independent of:
  - Programming model
  - Interprocess communication API
  - Operating system
  - Software component implementation language
- Multiple development systems: Solaris and Windows® NT 4.0
  - Java-based
- Multiple target systems
  - Currently RACE® and UNIX
  - IBM workstations and IBM SP2 under development at Maui HPCC
- Multiple communication APIs: MPI, PAS, ICS, POSIX

Talaris™ Productivity

Enables cooperating client tools to share centralized information about an application's software and hardware entities and relationships

- PeakWare™ for RACE® application builder with interprocessor communication and synchronization auto-code generator
- Performance trace profiling and analysis
- SuperVision debugger and performance monitor

Rapid interface validation - executable interface description

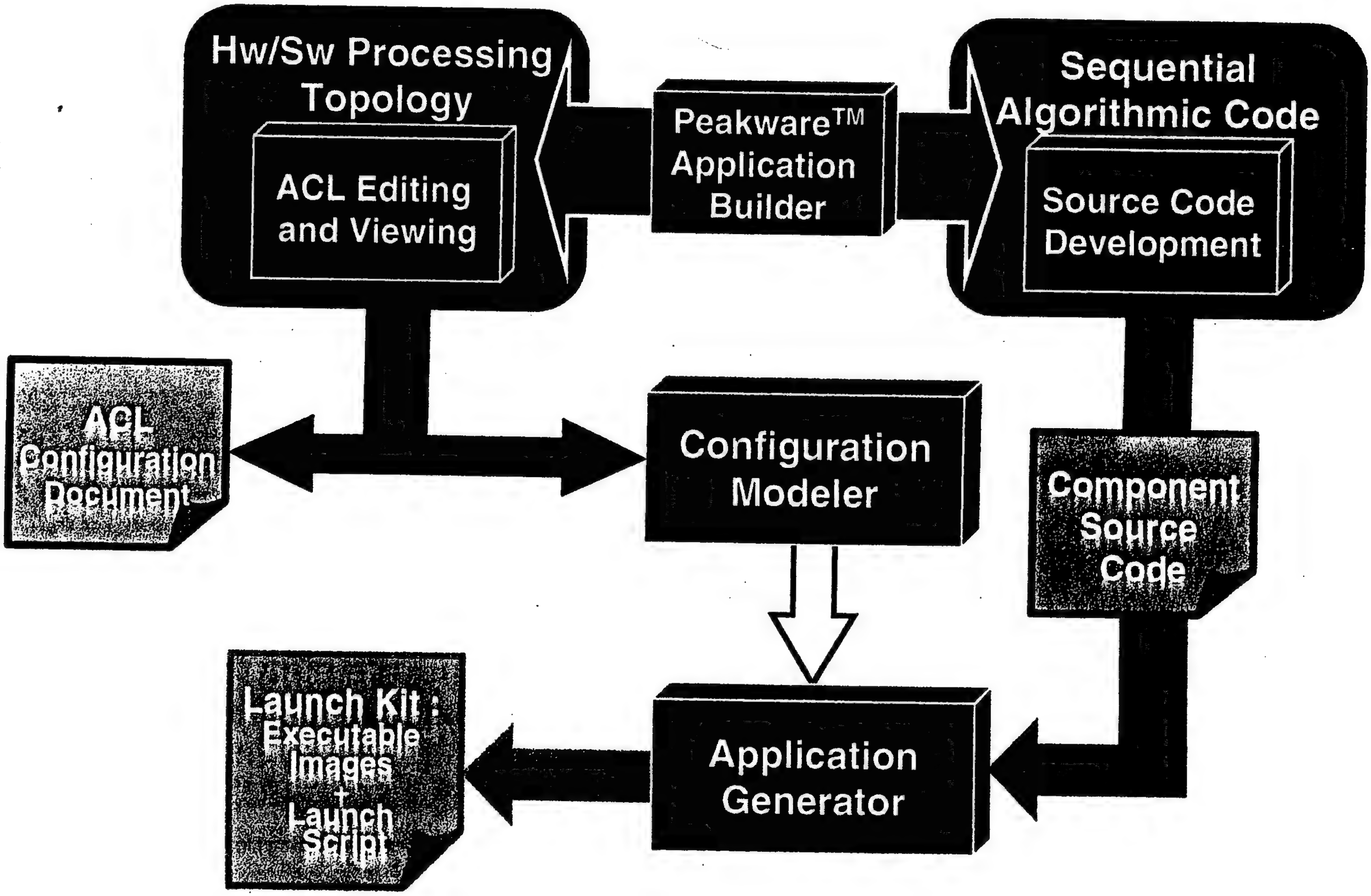
Imposes structure encouraging reusable component design

Eliminates initialization software, and reduces makefiles and shell scripts

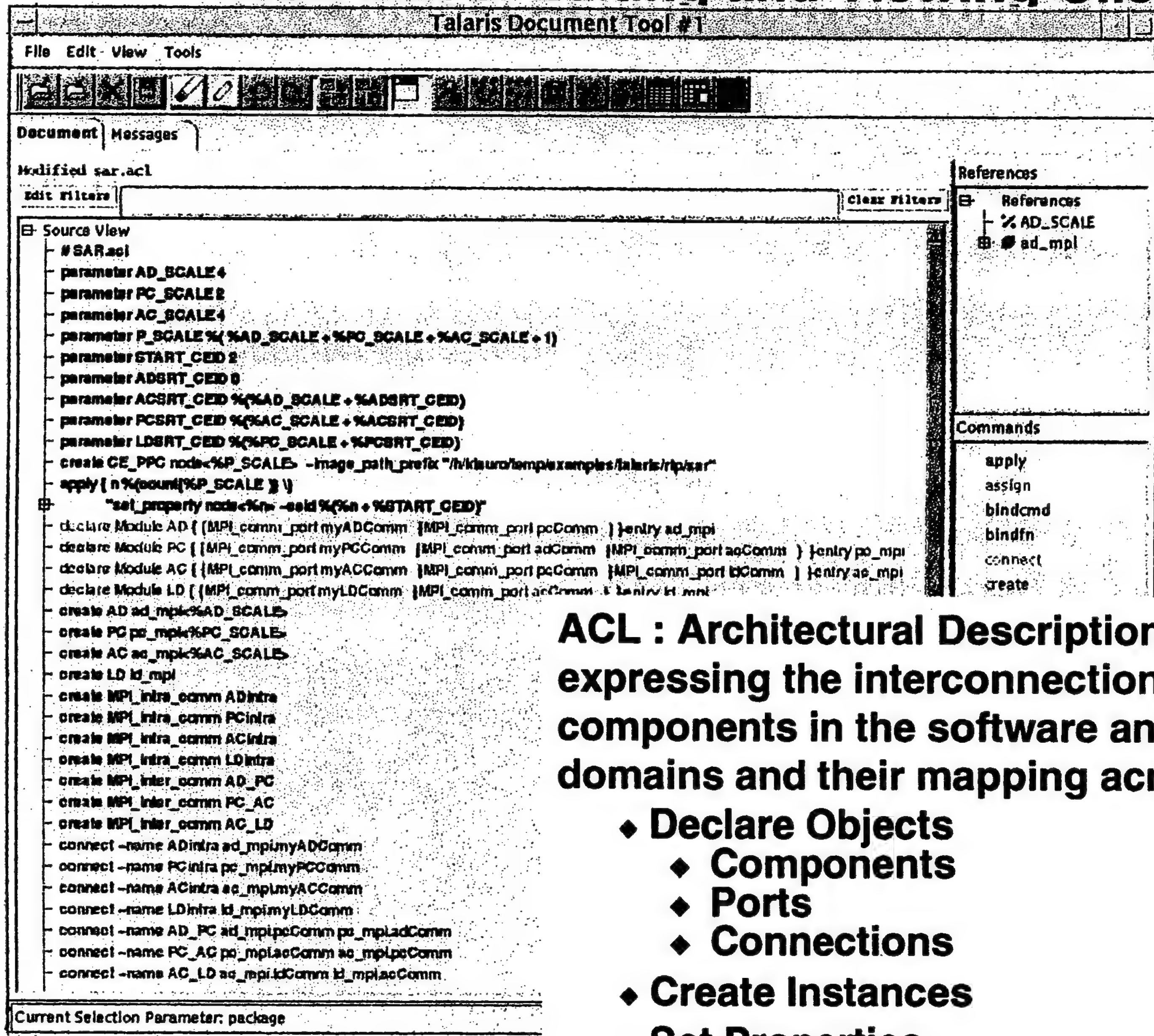
Talaris™ Development Environment

- An ACI application description is comprised of the interface definition of the software components, plus a description of the Hardware and Software processing topologies. It may be generated by a client tool or user-written.
- Component modules conforming to the interface definition are obtained from reusable software module libraries, or application-specific source code modules created manually or auto-generated. This module inventory provides a reusable code base for concurrent and future projects. At the beginning of the design cycle, modules can be modules stubs which allow early interface validation and application emulation. Overall processing topology can be modified without modifying source code
- The Generator receives a view from the Configuration Modeler describing the application's entities and relationships that represent the configuration of software and hardware components and the mapping of the software onto a target hardware configuration. The Generator analyzes and validates the application, provides immediate error checking, determines what runs where, devises a plan for initializing hardware modules and software connections, generates the C source files for Agent Module, Makefiles, and run scripts; links component modules and Agent Module to form generated executables, creates the Launch Script used later to initialize and start the application. The Launch Script and executables are stored as a Launch Kit for later execution.

Talaris™ Development Environment



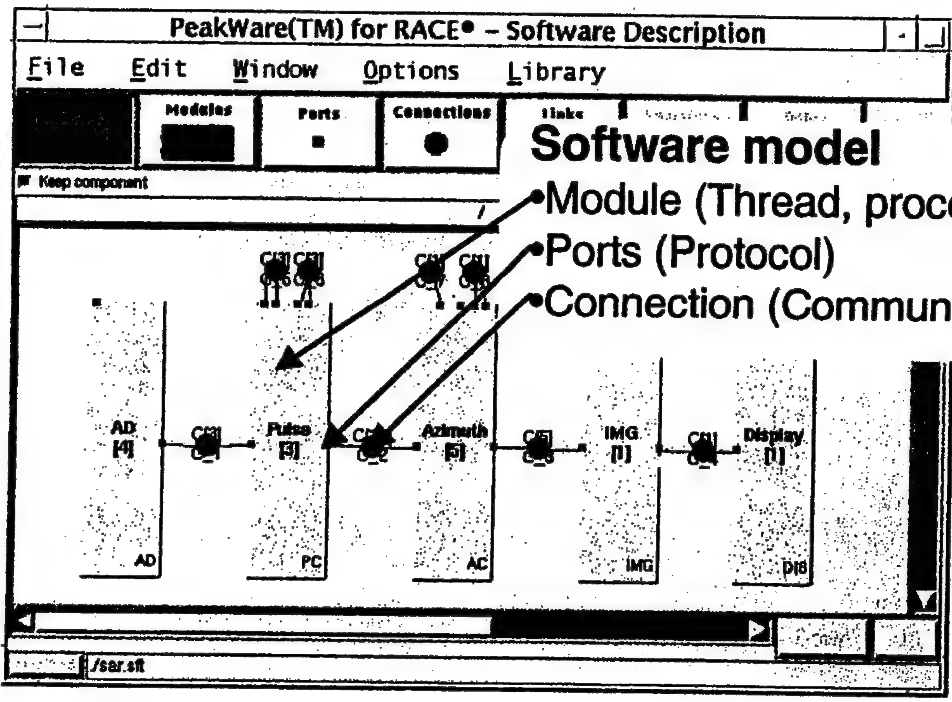
ACL Model Editing and Viewing Clients



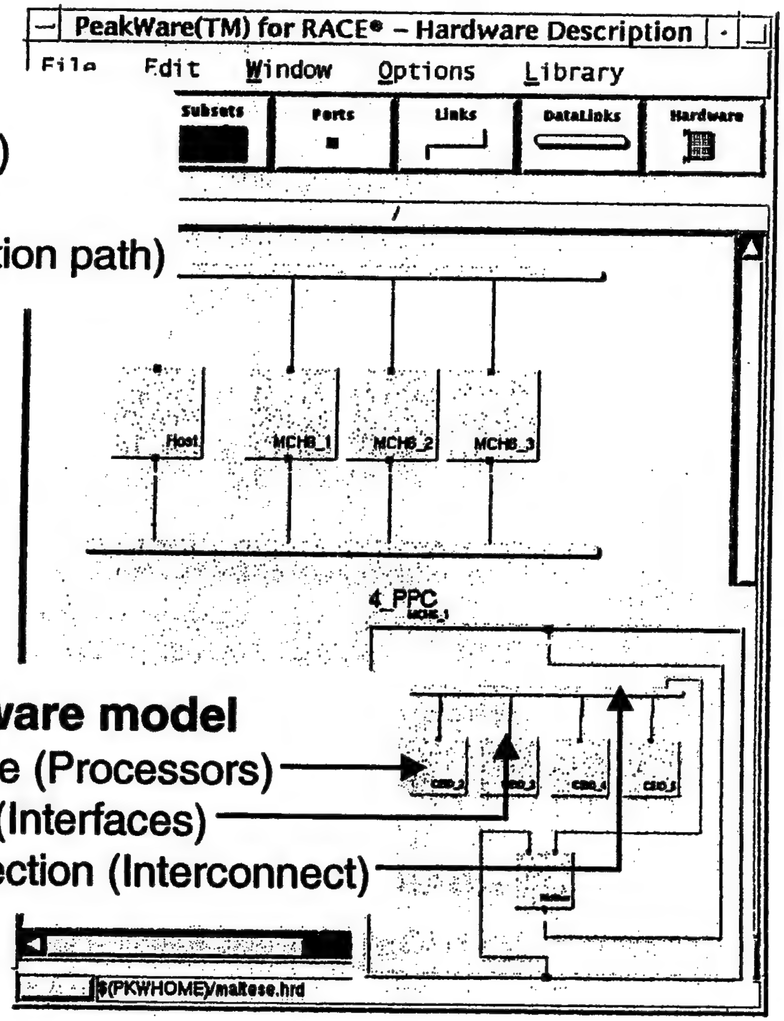
ACL : Architectural Description Language for expressing the interconnection of components in the software and hardware domains and their mapping across domains.

- **Declare Objects**
  - **Components**
  - **Ports**
  - **Connections**
- **Create Instances**
- **Set Properties**

PeakWare™ Application Builder Client



Software model
- •Module (Thread, process)
- •Ports (Protocol)
- •Connection (Communication path)



Hardware model
- •Module (Processors)
- •Ports (Interfaces)
- •Connection (Interconnect)

Generates code for:
- ◆Interprocessor communication and synchronization
- ◆Shared memory allocation

Talaris™ Runtime Environment

The Launcher orchestrates the global application, while individual agent models orchestrate thread processing locally.

The Launcher on the System CE

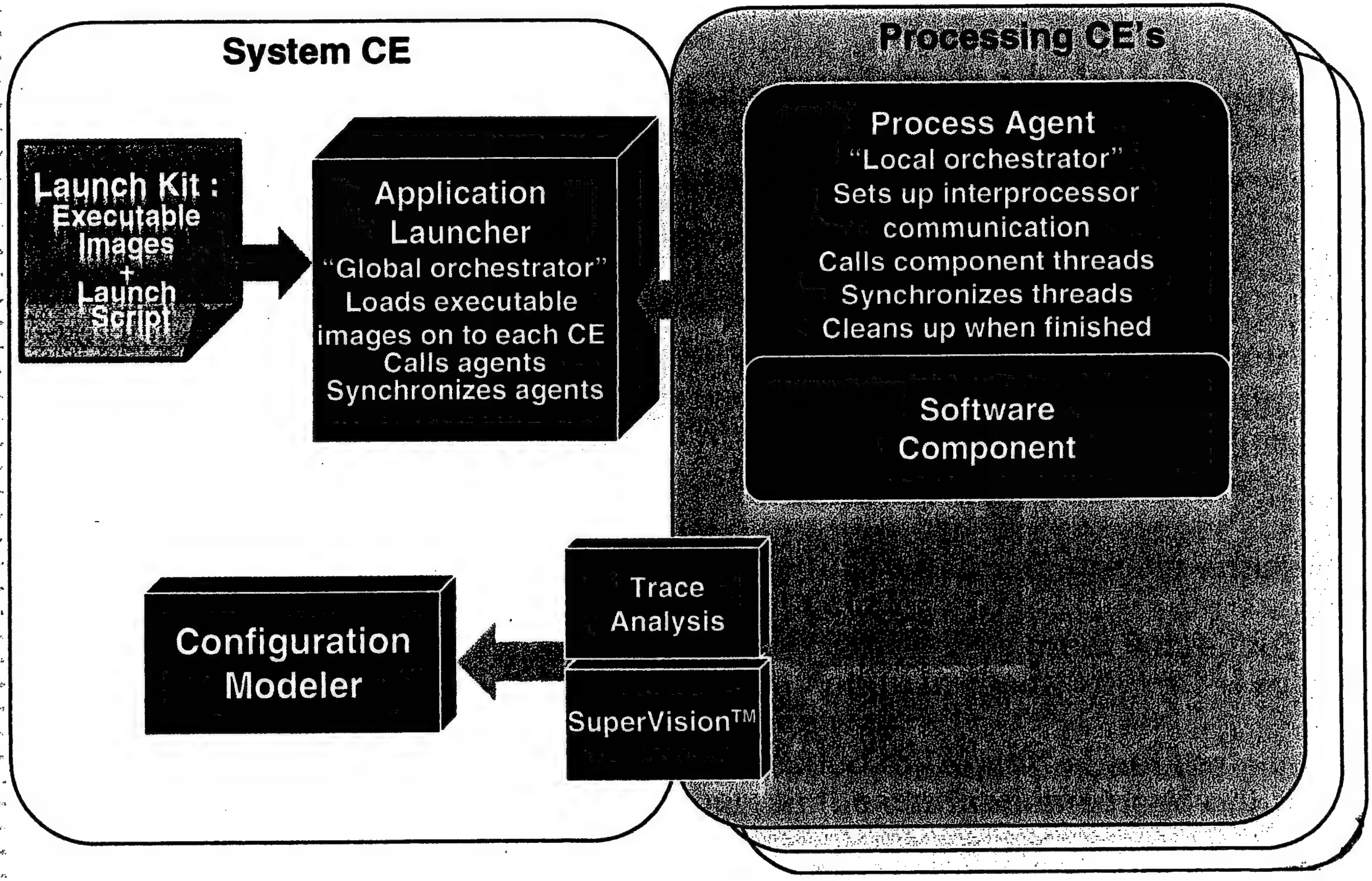
- Reads in the Launch kit, intializes the embedded hardware and generates individual agent scripts based on the launch script instructions.
- Loads executable images on to each processing CE, sets up global IPCs for agent synchronization, and spawns threads in processes, then gives a start message to the agents modules.
- Controls the progress of Agent modules using globally scoped "sync semaphores", based on information provided by the Agents on an ongoing basis.

The Agent on the processing CE's

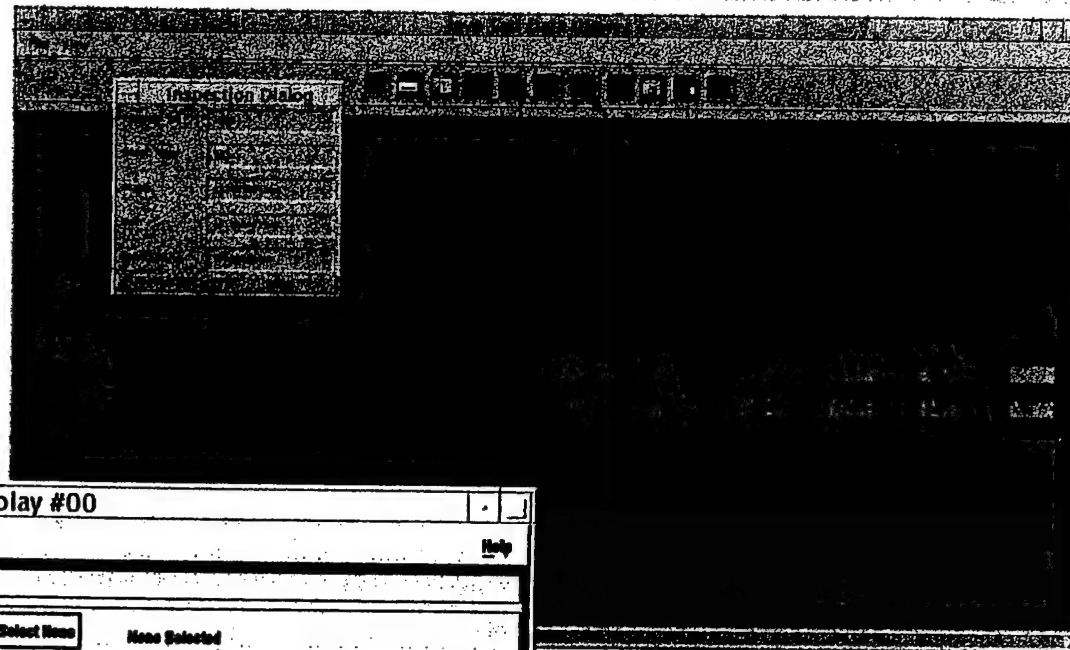
- Supplies the main() entry point for the process.
- Provides for IPC services setup and synchronizes threads/processes..
- Reports status back to Launcher
- When directed by the Launcher, starts the executing user modules.

The user can choose to execute the application detached from the Launcher.

Talaris™ Runtime Environment

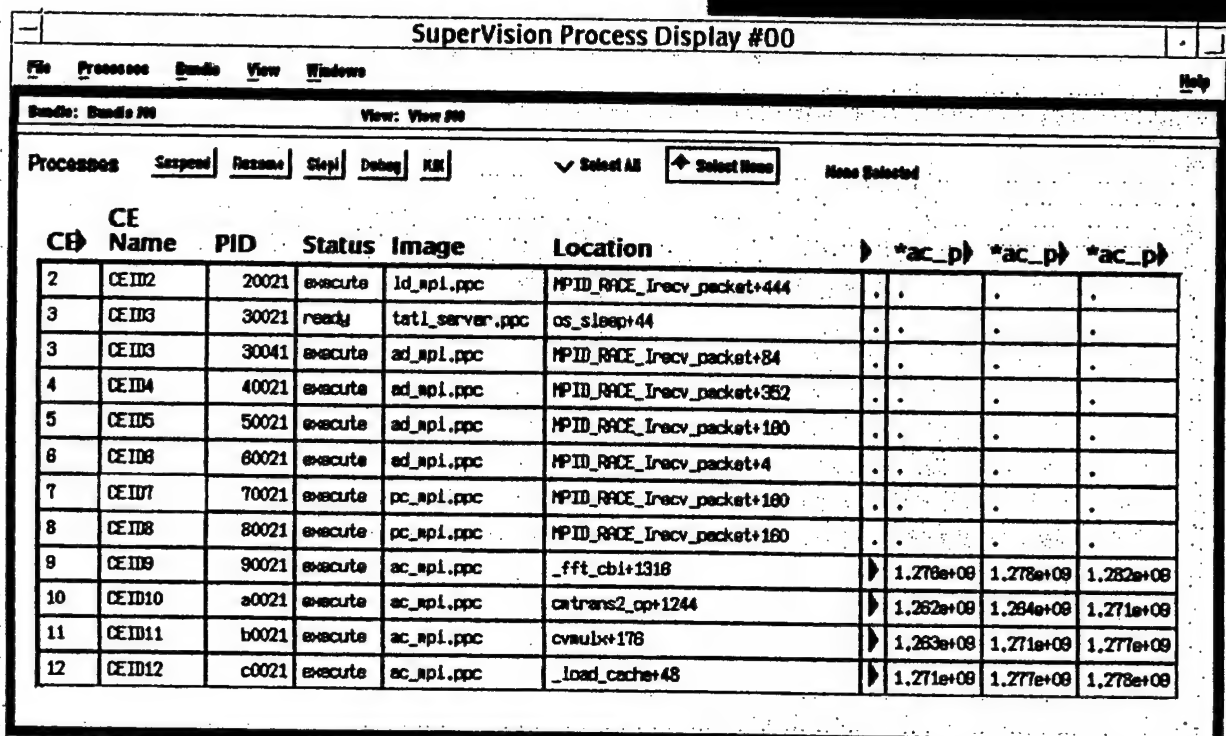


Debugging & Performance Clients



Event Tracing

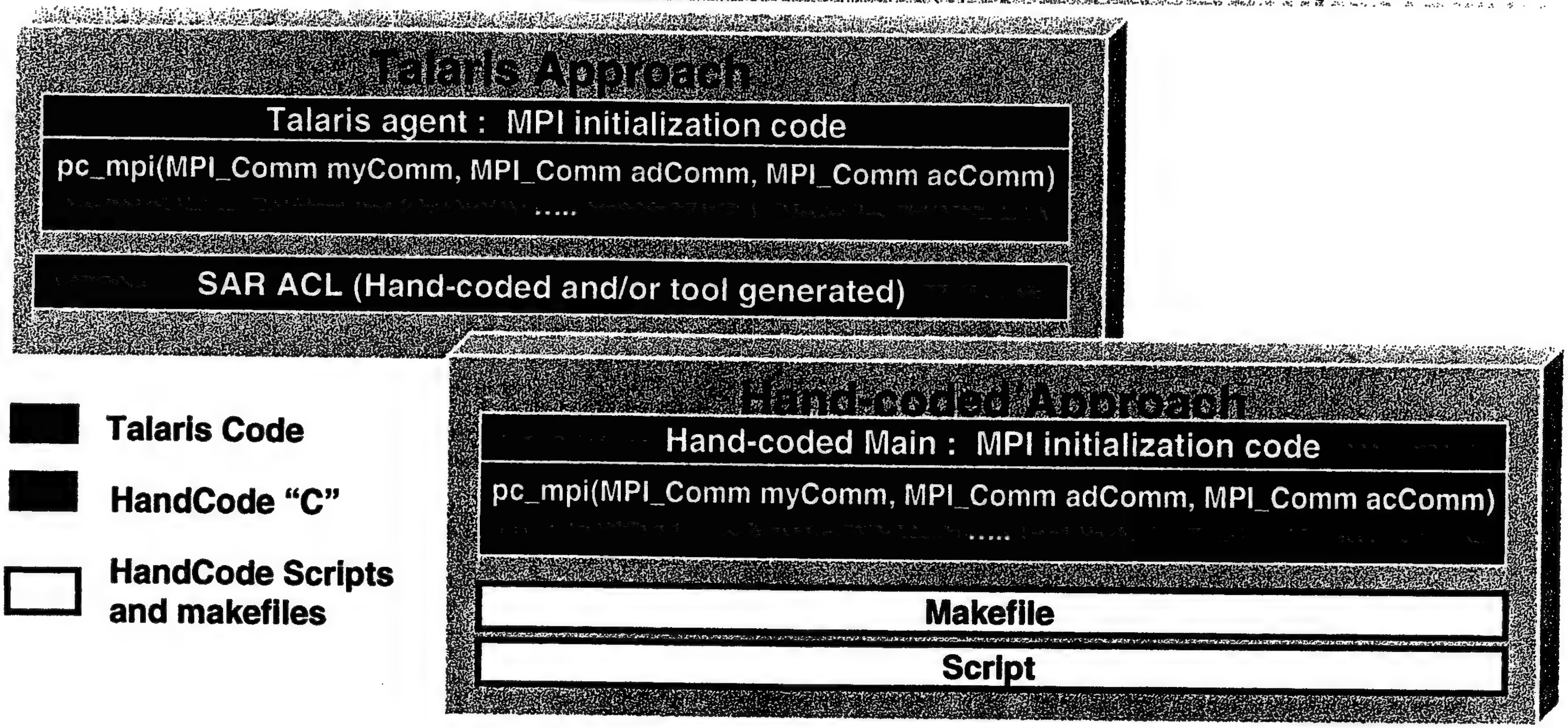
Post-mortem analysis supports accurate depiction of application execution efficiency for the detection of performance bottlenecks and sources of non-determinism.



Supervision™ Debugger & Performance Monitor

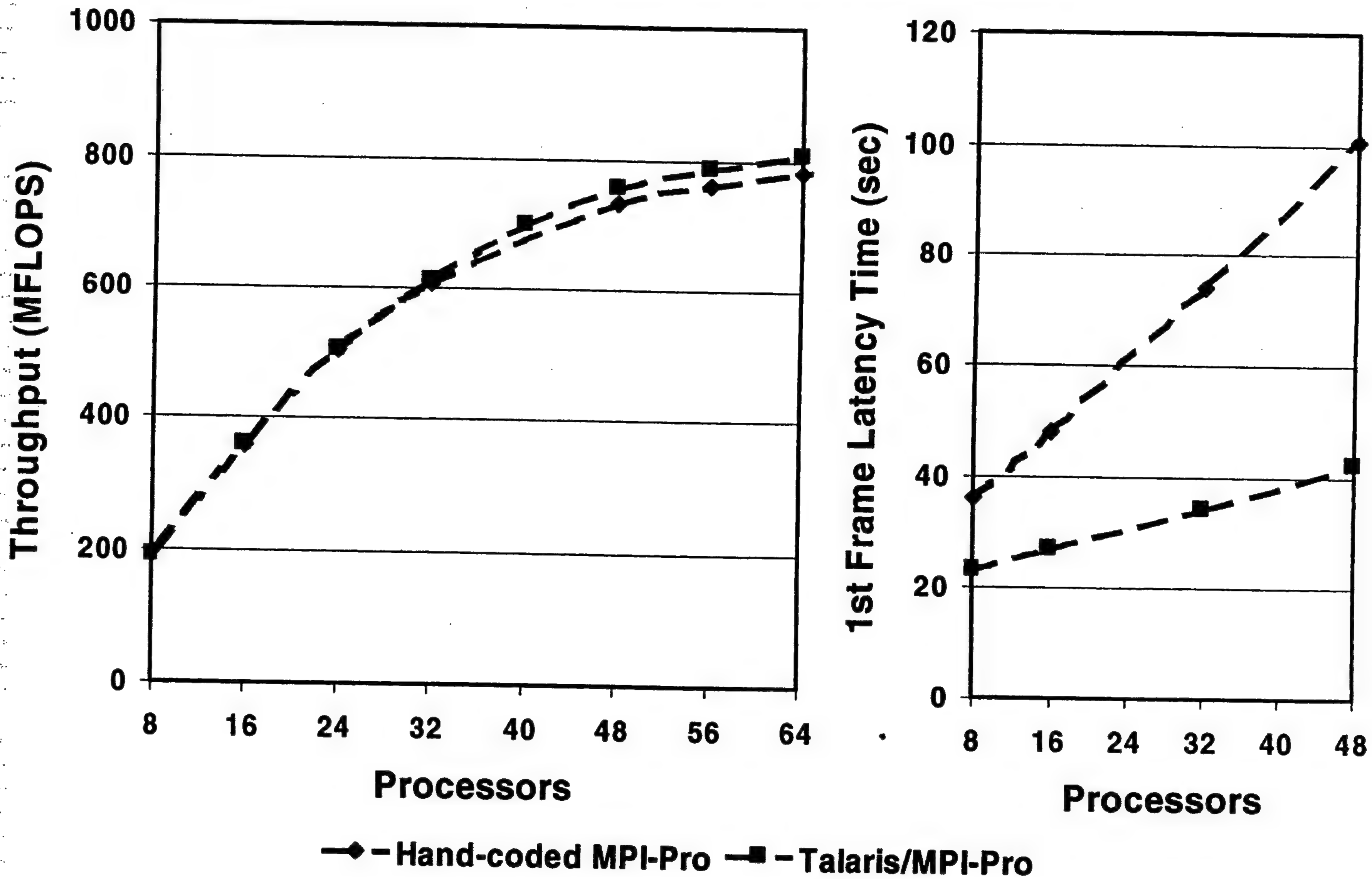
Monitor and control processes, and view global data on multiple processors in a heterogeneous multi-computer system

Synthetic Aperture Radar Application Code Comparison



- Portable software modules, or components, were developed using Message-Passing Interface (MPI) as the interprocessor communication API.
- These components were then used to develop Hand-coded and Talaris-supported applications.

Hand Coded / Talaris Performance Comparison
Sustained Throughput and 1st Frame Latency



Conclusion
Lessons Learned

Talaris is the middleware for the graphical application development product, PeakWare™ for RACE, developed by MATRA SYSTEMES & INFORMATION in collaboration with Mercury Computer Systems, Inc.

Talaris remains the focus of ongoing research.

<u>Application Development Lessons</u>

- Portability allows progress in an imperfect world! Talaris portability > 99%.
- Performance optimization is much easier if you can see where the time is being wasted. Talaris has an efficient application launcher, and runtime performance not impacted by process agent.
- Productivity Environment
  - Talaris enables previously disjoint clients to interact, greatly enhancing the utility of each and the development cycle.
  - A central repository of application information has great value, but developer wants selective views at any point in time.
  - Early interface validation smoothes the way.
  - The "mains" replaced by Talaris™ needed to "know" about the all the players in the full application, whereas the components need only to know about those to whom it communicates directly.

<u>Acknowledgements:</u> The Talaris™ environment was developed by Mercury Computer Systems and funded in part by the Defense Advanced Research Projects Agency project BAA97-07.

DISTRIBUTION LIST

addresses	number of copies
RALPH KOHLER AFRL/IFTC 26 ELECTRONICS PKWY ROME NY 13441-4514	2
MERCURY COMPUTER SYSTEMS, INC. 199 RIVERNECK ROAD CHELMSFORD MA 01824-2820	2
AFRL/IFOIL TECHNICAL LIBRARY 26 ELECTRONIC PKY ROME NY 13441-4514	1
ATTENTION: DTIC-OCC DEFENSE TECHNICAL INFO CENTER 8725 JOHN J. KINGMAN ROAD, STE 0944 FT. BELVOIR, VA 22060-6218	1
DEFENSE ADVANCED RESEARCH PROJECTS AGENCY 3701 NORTH FAIRFAX DRIVE ARLINGTON VA 22203-1714	1
ATTN: NAN PFRIMMER IIT RESEARCH INSTITUTE 201 MILL ST. ROME, NY 13440	1
AFIT ACADEMIC LIBRARY AFIT/LDR, 2950 P.STREET AREA B, BLDG 642 WRIGHT-PATTERSON AFB OH 45433-7765	1
ATTN: SMDC IM PL US ARMY SPACE & MISSILE DEF CMD P.O. BOX 1500 HUNTSVILLE AL 35807-3801	1

TECHNICAL LIBRARY D0274(PL-TS) 1
SPAWARSYSCEN
53560 HULL ST.
SAN DIEGO CA 92152-5001

COMMANDER, CODE 4TL000D 1
TECHNICAL LIBRARY, NAWC-WD
1 ADMINISTRATION CIRCLE
CHINA LAKE CA 93555-6100

CDR, US ARMY AVIATION & MISSILE CMD 2
REDSTONE SCIENTIFIC INFORMATION CTR
ATTN: AMSAM-RD-OB-R, (DOCUMENTS)
REDSTONE ARSENAL AL 35898-5000

REPORT LIBRARY 1
MS P364
LOS ALAMOS NATIONAL LABORATORY
LOS ALAMOS NM 87545

AFIWC/MSY 1
102 HALL BLVD, STE 315
SAN ANTONIO TX 78243-7016

USAF/AIR FORCE RESEARCH LABORATORY 1
AFRL/VSOSA(LIBRARY-BLDG 1103)
5 WRIGHT DRIVE
HANSCOM AFB MA 01731-3004

ATTN: EILEEN LADUKE/D460 1
MITRE CORPORATION
202 BURLINGTON RD
BEDFORD MA 01730

OUSD(P)/DTSA/DUTD 1
ATTN: PATRICK G. SULLIVAN, JR.
400 ARMY NAVY DRIVE
SUITE 300
ARLINGTON VA 22202

MISSION OF AFRL/INFORMATION DIRECTORATE (IF)

The advancement and application of information systems science and technology for aerospace command and control and its transition to air, space, and ground systems to meet customer needs in the areas of Global Awareness, Dynamic Planning and Execution, and Global Information Exchange is the focus of this AFRL organization. The directorate's areas of investigation include a broad spectrum of information and fusion, communication, collaborative environment and modeling and simulation, defensive information warfare, and intelligent information systems technologies.